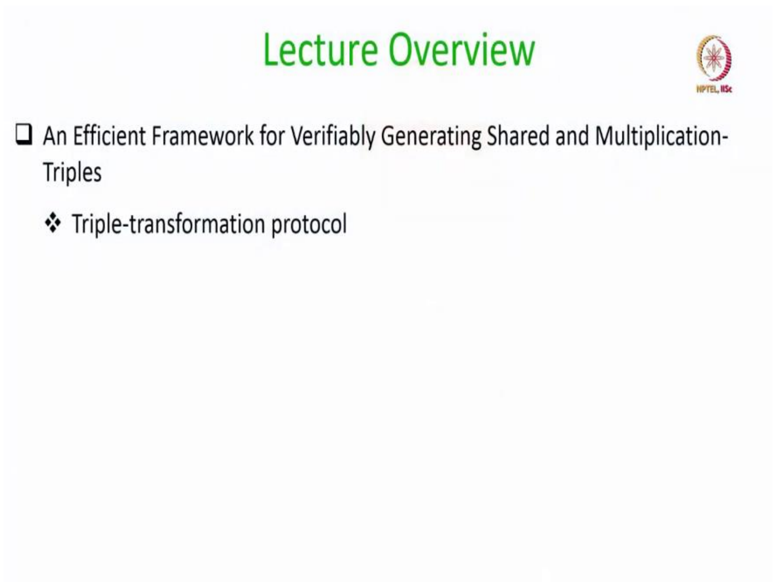
Secure Computation: Part II

Prof. Ashish Choudhury
Department of Computer Science and Engineering
Indian Institute of Science, Bengaluru

Lecture - 49
Generating Random Multiplication-Triples: II

(Refer Slide Time: 00:26)

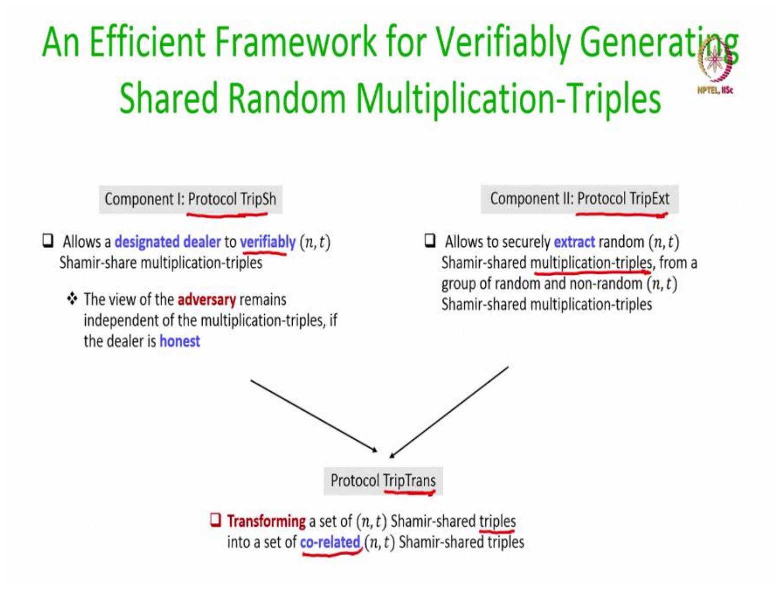


Lecture Overview

- ❑ An Efficient Framework for Verifiably Generating Shared and Multiplication-Triples
  - ❖ Triple-transformation protocol

Hello everyone. Welcome to this lecture. So, in this lecture we will continue our discussion regarding the efficient framework for generating the secret sharing of Random Multiplication Triplets, which we started discussing in the last lecture. And in this lecture, we will see a triple transformation protocol which will be useful for instantiating both the components of the above-mentioned framework.

(Refer Slide Time: 00:55)



An Efficient Framework for Verifiably Generating Shared Random Multiplication-Triples

Component I: Protocol TripSh

- ❑ Allows a **designated dealer** to **verifiably** (n, t) Shamir-share multiplication-triples
  - ❖ The view of the **adversary** remains independent of the multiplication-triples, if the dealer is **honest**

Component II: Protocol TripExt

- ❑ Allows to securely **extract** random (n, t) Shamir-shared multiplication-triples, from a group of random and non-random (n, t) Shamir-shared multiplication-triples

Protocol TripTrans

- ❑ **Transforming** a set of (n, t) Shamir-shared triples into a set of **co-related** (n, t) Shamir-shared triples

So, just to quickly recap the framework requires 2 protocols. The 1st protocol is the triple sharing protocol, which allows a dealer to secret share multiplication triples, where if the dealer is honest then its multiplication triples remain private. Unknown to the adversary and the verifiability guarantees that even if the dealer is corrupt, it has secret shared multiplication triplets and not arbitrary triplets.

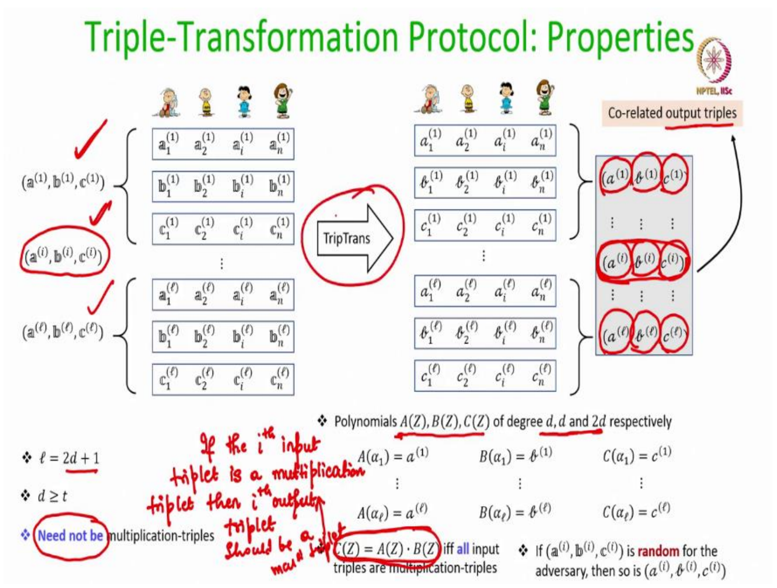
And the 2nd protocol is the triple extraction protocol, which takes a bunch of random and nonrandom secret shared multiplication triplets. Namely, there will be a set of multiplication triplets, each of which is secret shared. Some of these multiplication triplets will be known to the adversary, some of the multiplication triplets will be random from the viewpoint of the adversary.

The exact identity of the multiplication triplets, which are unknown to the adversary, will not be known without knowing which multiplication triplets are unknown for the adversary; this triple extraction procedure helps us to get a bunch of secret shared multiplication triplets all of which are guaranteed to be random from the viewpoint of the adversary.

Now, we want to instantiate this triple sharing protocol and triple extraction protocol, but before going into the instantiation we have to discuss another protocol called triple transformation protocol TripTrans, which is a common gadget used both for instantiating the triple sharing protocol as well as the triple extraction protocol. And as the name suggest triple transformation TripTrans. TripTrans here stands for triple transformation. What exactly is the transformation?

The transformation here is that it takes a bunch of secret shared triples, which may not have any relation among them, and then transforms them into a bunch of secret shared triples, which have a correlation. What exactly is the correlation? We will see very soon.

(Refer Slide Time: 03:37)


NPTEL, IISc
$(\mathbb{a}^{(1)}, \mathbb{b}^{(1)}, \mathbb{c}^{(1)})$
$(\mathbb{a}^{(i)}, \mathbb{b}^{(i)}, \mathbb{c}^{(i)})$
$(\mathbb{a}^{(\ell)}, \mathbb{b}^{(\ell)}, \mathbb{c}^{(\ell)})$
$\mathbb{a}_1^{(1)}$ $\mathbb{a}_2^{(1)}$ $\mathbb{a}_i^{(1)}$ $\mathbb{a}_n^{(1)}$
$\mathbb{b}_1^{(1)}$ $\mathbb{b}_2^{(1)}$ $\mathbb{b}_i^{(1)}$ $\mathbb{b}_n^{(1)}$
$\mathbb{c}_1^{(1)}$ $\mathbb{c}_2^{(1)}$ $\mathbb{c}_i^{(1)}$ $\mathbb{c}_n^{(1)}$
$\mathbb{a}_1^{(\ell)}$ $\mathbb{a}_2^{(\ell)}$ $\mathbb{a}_i^{(\ell)}$ $\mathbb{a}_n^{(\ell)}$
$\mathbb{b}_1^{(\ell)}$ $\mathbb{b}_2^{(\ell)}$ $\mathbb{b}_i^{(\ell)}$ $\mathbb{b}_n^{(\ell)}$
$\mathbb{c}_1^{(\ell)}$ $\mathbb{c}_2^{(\ell)}$ $\mathbb{c}_i^{(\ell)}$ $\mathbb{c}_n^{(\ell)}$
TripTrans
$a_1^{(1)}$ $a_2^{(1)}$ $a_i^{(1)}$ $a_n^{(1)}$
$b_1^{(1)}$ $b_2^{(1)}$ $b_i^{(1)}$ $b_n^{(1)}$
$c_1^{(1)}$ $c_2^{(1)}$ $c_i^{(1)}$ $c_n^{(1)}$
$a_1^{(\ell)}$ $a_2^{(\ell)}$ $a_i^{(\ell)}$ $a_n^{(\ell)}$
$b_1^{(\ell)}$ $b_2^{(\ell)}$ $b_i^{(\ell)}$ $b_n^{(\ell)}$
$c_1^{(\ell)}$ $c_2^{(\ell)}$ $c_i^{(\ell)}$ $c_n^{(\ell)}$
$(a^{(1)}, b^{(1)}, c^{(1)})$
$(a^{(i)}, b^{(i)}, c^{(i)})$
$(a^{(\ell)}, b^{(\ell)}, c^{(\ell)})$
Co-related output triples
If the ith input triplet is a multiplication triplet then ith output triplet should be a mult triplet

Triple-Transformation Protocol: Properties

- ❖ $\ell = 2d + 1$
- ❖ $d \geq t$
- ❖ Need not be multiplication-triples
- ❖ Polynomials $A(Z), B(Z), C(Z)$ of degree d, d and $2d$ respectively

$A(\alpha_1) = a^{(1)}$ $B(\alpha_1) = b^{(1)}$ $C(\alpha_1) = c^{(1)}$

$\vdots$

$A(\alpha_\ell) = a^{(\ell)}$ $B(\alpha_\ell) = b^{(\ell)}$ $C(\alpha_\ell) = c^{(\ell)}$

- ❖ $C(Z) = A(Z) \cdot B(Z)$ iff all input triples are multiplication-triples
- ❖ If $(\mathbb{a}^{(i)}, \mathbb{b}^{(i)}, \mathbb{c}^{(i)})$ is random for the adversary, then so is $(a^{(i)}, b^{(i)}, c^{(i)})$

So, this triple transformation protocol takes a set of n number of secret shared triplets, where ℓ will be of the form $2d + 1$ and d will be at least t. And I stressed that these triplets need not be multiplication triplets. In fact, it could be the pause, it could be the case that none of these input triplets is a multiplication triplet or it could be the case that all but one among these ℓ triplets is a non-multiplication triplet.

So, it could be any possibility had and there is absolutely no relationship among this ℓ triplets. They are just ℓ arbitrary triplets, which are secret shared. Now, this triple transformation protocol, transforms this input secret shared triplets into output secret shared triplets. So, the number of output triplets is same as the number of input triplets, but now the output triplets are no longer unrelated. They have some correlation among them ok.

So, before going into the correlation, you can see here that I have used different fonts to represent the input triplets and the output triplets. Just to avoid confusion here. So, what is the correlation which we want among the output triplets? The correlation is the following: we want that at the end of the protocol there should exist 3 polynomials and A polynomial, a B polynomial, and a C polynomial of degree d, d and $2d$ respectively, such that all the following properties hold.

The first property is that all the a components of the output triplets, they should lie on the A polynomial. So, if I consider the a component of the first output triplet, the a component of the second output triplet, and the a component of the ℓth sorry all of them should lie on my A polynomial, where the degree of the A polynomial is d.

In the same way I require the b component of all the output triplets to lie on the B polynomial. And in the same way I would like the c component of all the output triplets to lie on the C polynomial. Of course, this output triplets have to be available in the secret shared fashion, where the degree of sharing should be t. That is the first correlation.

The second correlation that we want here is that the C polynomial should be equal to the product of the A and B polynomial, if and only if all the input triplets are multiplication triplets, stated in a different way. What I require here is that if the ith input triplet is a multiplication triplet, then ith output triplet should be a multiplication triplet and this is an if and only if condition.

That means I want the property to hold in the other direction as well; that means, if the ith output triplet is a multiplication triplet, then the ith input triplet was also a multiplication triplet. That automatically guarantees that if all the input triplets are multiplication triplets; the first, the second, the ith, and the ℓth, then automatically all the output triplets will be multiplication triplets. As a result of that since, the degrees of A, B, and C polynomials are d, d and $2d$ respectively, it will automatically be guaranteed that the C polynomial is the product of the A polynomial and B polynomial.

Because the a component of all the output triplets b components of all the output triplets and c components of all the output triplets, they are distinct points on the A, B, and C polynomials respectively. So, that is a very powerful correlation relationship. So, even if my inputs are arbitrary triplets, I have tied them together through this triple transformation process by ensuring that I transform the input triplets into output triplets in such a way that the output triplets now constitute distinct points on the $A, B,$ and C polynomials.

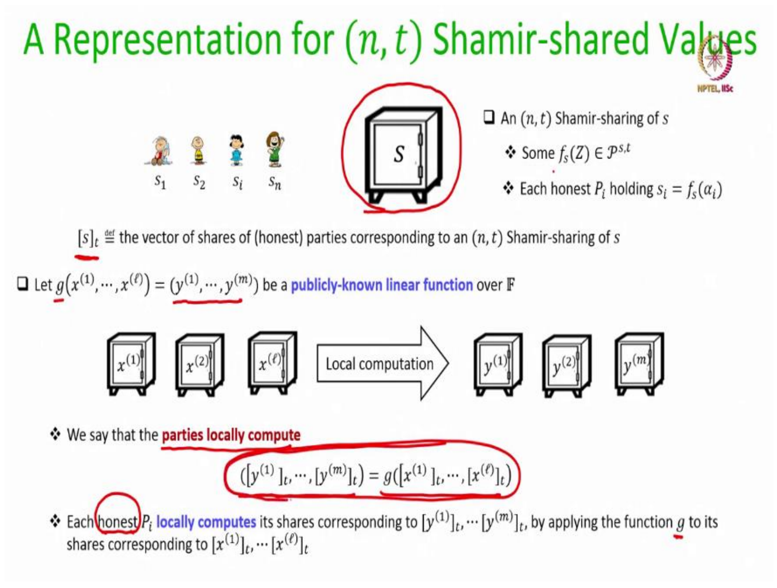
However, you see the degrees of the A and B polynomials are d whereas, the degree of the C polynomial is $2d$ and the second correlation that I want here is that if all the input triplets are multiplication triplets, then all the output triplets will be multiplication triplets and that will further guarantee that the C polynomial will be the product of A and B polynomial. However, if any of the input triplet is not a multiplication triplet, then the C polynomial will not be the product of A and B polynomial.

Say for instance, the ith triplet is not a multiplication triplet, then the ith output triplet should also be should also not be a multiplication triplet and since, the ith output triplet constitute distinct points on the $A, B,$ and C polynomial. We know now that there is at least one point on the $A, B,$ and C polynomial, which violates this multiplicative relationship. So, that is the second correlation, which we require from the triple transformation protocol.

And the third property which we require from the triple transformation protocol is that if the ith input triplet was random for the adversary. Of course, adversary will be knowing t shares for all the input triplets as well as for the output triplets because everything here is secret shared. But it could be possible that the ith input triplet was random for the adversary; it only knows t shares for the ith input triplet. If that is the case, then we would require that the privacy for the ith output triplet is also maintained that is a third requirement.

So, that these are the 3 properties we require from the triple transformation protocol. Later on, we will see that assuming we have this triple transformation protocol, how we can instantiate the 2 components of our efficient framework for the pre processing phase protocol. So, now we will see in this lecture how to instantiate the triple transformation protocol.

(Refer Slide Time: 12:33)



A Representation for (n, t) Shamir-shared Values

NPTEL, IISc
s_1
s_2
s_i
s_n
s

- An (n, t) Shamir-sharing of s
  - Some $f_s(Z) \in \mathcal{P}^{s,t}$
  - Each honest P_i holding $s_i = f_s(\alpha_i)$

$[s]_t \overset{\text{def}}{=}$ the vector of shares of (honest) parties corresponding to an (n, t) Shamir-sharing of s

- Let $g(x^{(1)}, \cdots, x^{(\ell)}) = (y^{(1)}, \cdots, y^{(m)})$ be a publicly-known linear function over $\mathbb{F}$

$x^{(1)}$
$x^{(2)}$
$x^{(\ell)}$
Local computation
$y^{(1)}$
$y^{(2)}$
$y^{(m)}$

- We say that the parties locally compute

$$([y^{(1)}]_t, \cdots, [y^{(m)}]_t) = g([x^{(1)}]_t, \cdots, [x^{(\ell)}]_t)$$

- Each honest P_i locally computes its shares corresponding to $[y^{(1)}]_t, \cdots [y^{(m)}]_t$, by applying the function g to its shares corresponding to $[x^{(1)}]_t, \cdots [x^{(\ell)}]_t$

Before going into the instantiation of the triple transformation protocol, for the purpose of representation, I will introduce here a notation for representing Shamir-secret shared values. So, imagine there is a value s from the field, which is secret shared as per the Shamir-secret sharing scheme through a t degree polynomial with every party holding a share for this secret; that means there exists some t degree polynomial whose constant term is s and ith party holds the value of the Shamir-sharing polynomial at α_i.

If that is the setting, then to represent that setting I will use this box representation. Why this box representation? This box representation basically signifies here that there is a value s which none of the party knows. Of course, the dealer who would have secret shared the value s would know the value s.

But if the dealer is honest from the viewpoint of the corrupt parties, there is a value in this box which is not known to the adversary. Adversary has only t shares for the value which is kept inside the box. And we will also use this notation $[s]_t$ to denote the vector of shares corresponding to a secret sharing of s. Now, we already know the linearity property of Shamir-secret sharing.

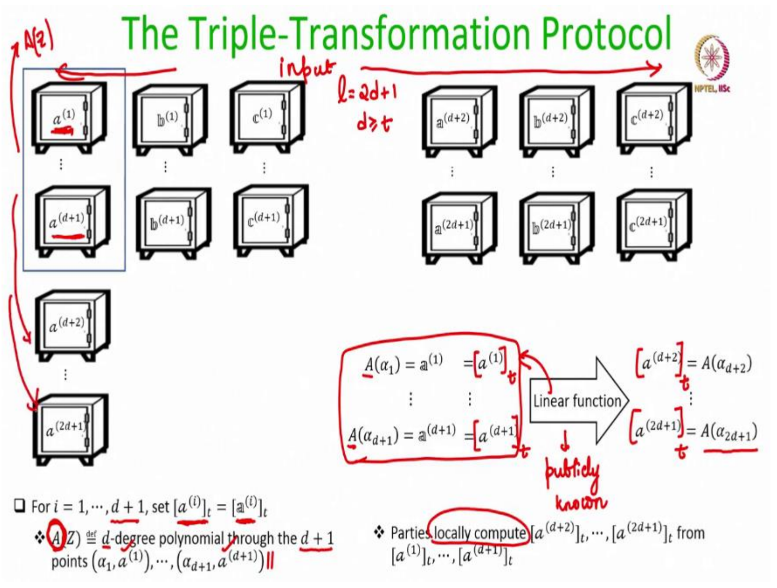
So, we know that if the if there is some publicly non-linear function g over the field and if the inputs for this function g are secret shared as per Shamir-secret sharing, then without any interaction the parties can locally compute their respective shares corresponding to the output of the function g. So, that will be represented pictorially by the following representation.

So, you have now the inputs for this function g in the box representation. That means, for x_1 each party has is its share, for x_2 each party has its share, for x_ℓ each party has its share, and now, the parties can locally apply the function g on its respective share of $x_1, x_2, \dots, x_\ell$ which will help every party to obtain its share for namely the output $y_1, y_2, \dots, y_m$ for the function g.

So, this whole process I will pictorially represent like this, and I will also use statements like parties locally compute the output of the function g by writing this expression.

So, whenever I write this expression; that means, I want to say that every party P_i, every honest party to be more specific, because corrupt party a maliciously corrupt party P_i may not follow protocol instructions, but every honest party P_i it takes its shares of $x_1, x_2, \dots, x_\ell$ corresponding to the secret sharing of $x_1, x_2, \dots, x_\ell$ applies the function g and gets its shares corresponding to $y_1, y_2, \dots, y_m$ that is what I mean whenever I write this expression.

(Refer Slide Time: 16:17)


The Triple-Transformation Protocol

$A(\alpha_1) = a^{(1)} = [a^{(1)}]_t$

$A(\alpha_{d+1}) = a^{(d+1)} = [a^{(d+1)}]_t$

$[a^{(d+2)}]_t = A(\alpha_{d+2})$

$[a^{(2d+1)}]_t = A(\alpha_{2d+1})$

❑ For $i = 1, \cdots, d+1$, set $[a^{(i)}]_t = [a^{(i)}]_t$

❖ $A(Z) \overset{\text{def}}{=} d$-degree polynomial through the $d+1$ points $(\alpha_1, a^{(1)}), \cdots, (\alpha_{d+1}, a^{(d+1)})$ ||

❖ Parties locally compute $[a^{(d+2)}]_t, \cdots, [a^{(2d+1)}]_t$ from $[a^{(1)}]_t, \cdots, [a^{(d+1)}]_t$

So, now coming to the triple transformation protocol; so, this is your input. What exactly is the input here? The input is a bunch of ℓ number of secret shared triplets. They need not be multiplication triplets. I stress here where $\ell = 2d + 1$ and $d \geq t$ and they may not exist any correlation among these secret shared triplets. Now, the steps of the triple transformation protocol are the following: the first $d+1$, for the first $d+1$ output triplets, we set the a component to be the same as the input itself.

So, I take for I equal to $d+1$ the a component of the input triplets and I consider those triplets to be the first $d+1$ output triplets. So, it is basically just renaming that nothing else. So, whatever the shares for the first $d+1$ triplets parties have, they take they focus on the shares corresponding to the a components and those shares are only considered as the shares corresponding to the a component of the first $d+1$ output triplets.

So, in terms of box representation you can imagine that whatever shares the parties have for this $a^{(1)}$. Namely, the a component of the first triplet those shares are retained as the shares for the a component of the first output triplets and like that, whatever shares parties have for the a component of the $d+1$th input triplet, those shares only are retained as the shares for the a component of the $d+1$th output triplets.

Now, the A polynomial is defined to be the unique d degree polynomial passing through the $d+1$ distinct points, which are written down here; namely we assume that let A be the d degree polynomial over which the first $d+1$ a components of the output over which that a component of the first $d+1$ output triplets lie. And this A polynomial is a well-defined polynomial. Why is it a well-defined polynomial? Because its degree is d.

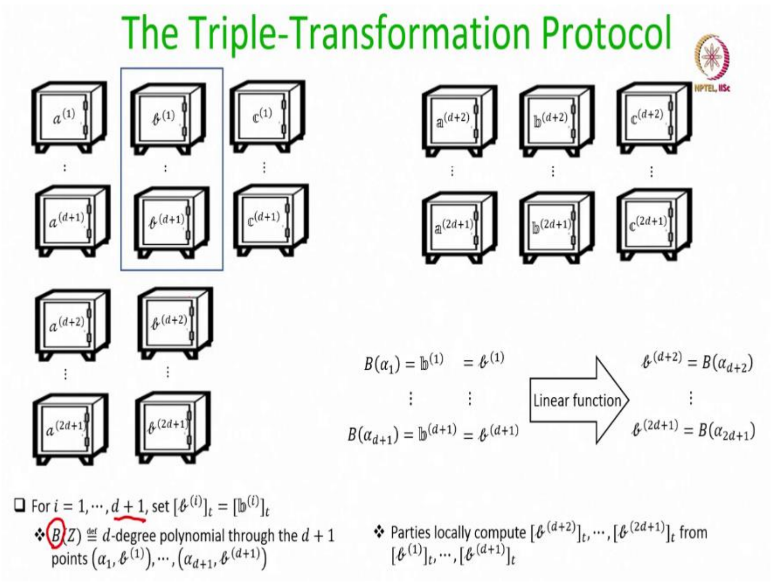
So, any d degree polynomial can be uniquely defined if I set or if I fix $d+1$ points on that polynomial. So, what are the $d+1$ points which I have fixed for the A polynomial? Those points are basically the a components of the first output triplets, which are the same as the a components for the first $d+1$ input triplets. I stress here that no one will know the value of this A polynomial; that means, none of the coefficients of a here is known to anyone.

Because every, because the points on this A polynomial; namely the a component of the first triplet, the a component of the second triplet, the a component of the $d+1$th output triplet, all of them are secret shared. As a result of that the coefficients of the polynomial A are also secret shared here. So, this is the way I have set my A polynomial. And now, I know that if I want to compute some new points on this A polynomial, namely the value of the polynomial at α_{d+2}, the value of the A polynomial at α_{d+3}, and the value of the A polynomial at α_{2d+1}.

Then these values are linear function of the first $d+1$ points on my A polynomial and this linear function is publicly known namely it is the Lagrange interpolation function. And we know that secret sharing satisfies the linearity property. So, we already have the secret sharing of the first point on the A polynomial. We already have the secret sharing of the second point on the A polynomial and we already have the secret sharing of the $d+1$th point on the A polynomial.

So, if I apply this linear function on those secret sharing, then that will enable us to get the secret sharing of the next points on the semi polynomial without doing any operation. So, as a result we can say that the parties now locally compute the a component of the remaining output triplets. That simply requires applying the linear function. So, you have fixed the A polynomial here in a secret shared fashion, apply the linear function and get this done.

(Refer Slide Time: 22:02)



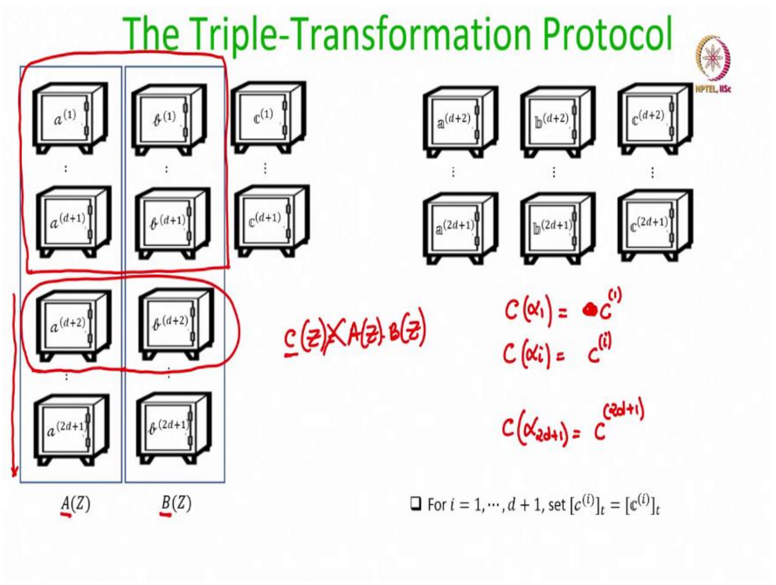
The Triple-Transformation Protocol

- For $i = 1, \cdots, d+1$, set $[b^{(i)}]_t = [b^{(i)}]_t$
  - ❖ $B(Z) \overset{\text{def}}{=} d$-degree polynomial through the $d+1$ points $(\alpha_1, b^{(1)}), \cdots, (\alpha_{d+1}, b^{(d+1)})$
  - ❖ Parties locally compute $[b^{(d+2)}]_t, \cdots, [b^{(2d+1)}]_t$ from $[b^{(1)}]_t, \cdots, [b^{(d+1)}]_t$

That is how the input that is how the a component of the input triplets are transformed into the a components of the output triplets. The same process we do even for the b components of the triplet namely, we take the first $d+1$. We take the b component of the first $d+1$ triplets, each of which is secret shared, and we consider them to be the b component of the first $d+1$ output triplets. And then, we define the unique d degree B polynomial passing through the b component of the first $d+1$ output triplets.

And then, we can apply the Lagrange linear function and get the next bunch of points on this defined B polynomial. Since everything is now linear function that linear function can be now applied on the secret sharing of the points on the B polynomial and we can get the transformed b components for the output triplets.

(Refer Slide Time: 23:18)


The Triple-Transformation Protocol

- For $i = 1, \cdots, d+1$, set $[c^{(i)}]_t = [\mathbb{c}^{(i)}]_t$

So, our A polynomial is defined, our B polynomial is defined; now our goal is to define the C polynomial. Now, you might be tempting to do the following. You might say that why we are anyhow given $2d + 1$ number of secret shared c components. Why cannot I define the C polynomial to be the following? That let C to be the $2d$ degree unique polynomial, which when evaluated at alpha 1 gives you the c component of the first triplet, when evaluated at α_i, gives you the c component of the ith triplet and when evaluated at alpha of $2d + 1$ gives you the c component of the last input triplet.

If I do this then you will say that I have now an A polynomial of degree d, B polynomial of degree d, and C polynomial of degree $2d$ fine. One of the requirements of the triple transformation protocol is achieved, but if I do if I define my C polynomial like this, then it is not guaranteed that the C polynomial is the product of the A polynomial and B polynomial, if all the input triplets are multiplication triplets, no.

Because, if you see here the way I have defined the A polynomial and B polynomial is as follows. The A polynomial is defined by fixing the first $d + 1$ points here. They are basically the first they are basically the a components of the first $d + 1$ input triplets, but the remaining points on the A polynomial, they are not same as the a components of the remaining input triplets. They are basically now new points on the A polynomial.

In the same way the B polynomial is defined by fixing or by setting the first $d+1$ points to be the b components of the first $d+1$ triplets, but the next bunch of d points on the B polynomial. They are absolutely different from the b component of the remaining input triplets, but for the C polynomial I am taking all the points on the C polynomial to be the c components of all the input triplets.

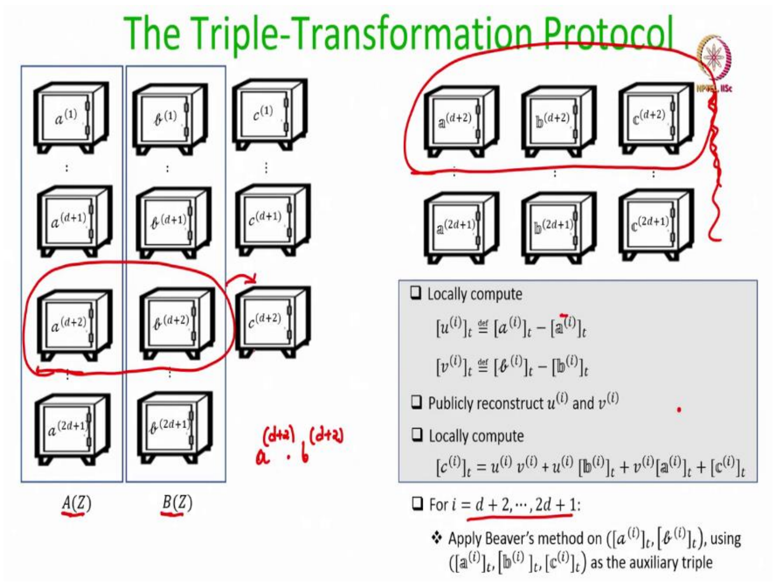
So, what can go wrong here is that for the first $d+1$ values, C polynomial evaluated at α_i will be same as the A polynomial at evaluated at α_i times B polynomial evaluated at α_i, if the first $d+1$ triplets are multiplication triplets fine. But for the remaining d values $C(\alpha_i)$ may not be equal to the product of $A(\alpha_i)$ and $B(\alpha_i)$, because the next bunch of points on the A and B polynomials they are different. Well, they are not different, they may be different from the a and b components of the remaining d number of input triplets.

So, that is why we cannot define the C polynomial like this. We have to do something else to ensure that the second property, which we require from the transformation is achieved. So, how do we define the C polynomial. So, we partially define the C polynomial first by fixing the $d+1$ points on the C polynomial to be the c components on the by fixing the by fixing the first $d+1$ points and the C polynomial to be the c components of the first $d+1$ input triplets; as we have done for the A and B polynomial.

So, so right now I have only $d+1$ points on the C polynomial fixed, but I want the C polynomial to be a $2d$ degree polynomial. So, to define a $2d$ degree polynomial I need d more points on the C polynomial; such that those points should be the product of the a components and the b components of the output triplet, if the a component and the b component and the c component were constituting a multiplication triplet.

So, how do I do that? So, for that for the remaining d points on the C polynomial, I obtain them by computing a secret sharing of the product of the a and b component by using the input triplets, which I have not yet touched and for that I apply the Beaver's method.

(Refer Slide Time: 28:50)



The Triple-Transformation Protocol

$a^{(1)}$ $\ell^{(1)}$ $c^{(1)}$
$a^{(d+1)}$ $\ell^{(d+1)}$ $c^{(d+1)}$
$a^{(d+2)}$ $\ell^{(d+2)}$ $c^{(d+2)}$
$a^{(2d+1)}$ $\ell^{(2d+1)}$
$A(Z)$
$B(Z)$
$\mathbb{a}^{(d+2)}$ $\mathbb{b}^{(d+2)}$ $\mathbb{c}^{(d+2)}$
$\mathbb{a}^{(2d+1)}$ $\mathbb{b}^{(2d+1)}$ $\mathbb{c}^{(2d+1)}$
$a^{(d+2)} \cdot \ell^{(d+2)}$

- ❑ Locally compute

  $[u^{(i)}]_t \overset{\text{def}}{=} [a^{(i)}]_t - [\mathbb{a}^{(i)}]_t$

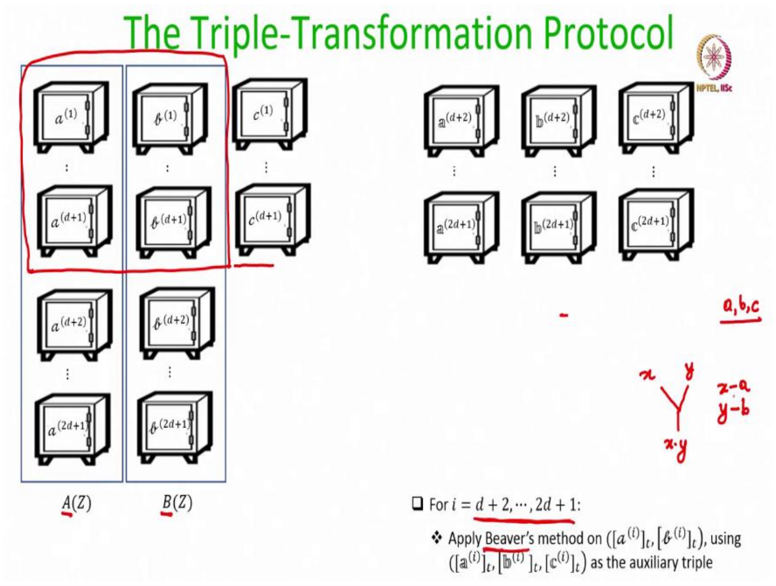
  $[v^{(i)}]_t \overset{\text{def}}{=} [\ell^{(i)}]_t - [\mathbb{b}^{(i)}]_t$

- ❑ Publicly reconstruct $u^{(i)}$ and $v^{(i)}$
- ❑ Locally compute

  $[c^{(i)}]_t = u^{(i)}\, v^{(i)} + u^{(i)}\, [\mathbb{b}^{(i)}]_t + v^{(i)}[\mathbb{a}^{(i)}]_t + [\mathbb{c}^{(i)}]_t$

- ❑ For $i = d+2, \cdots, 2d+1$:
  - ❖ Apply Beaver's method on $([a^{(i)}]_t, [\ell^{(i)}]_t)$, using $([\mathbb{a}^{(i)}]_t, [\mathbb{b}^{(i)}]_t, [\mathbb{c}^{(i)}]_t)$ as the auxiliary triple

(Refer Slide Time: 28:56)



The Triple-Transformation Protocol

$a^{(1)}$ $\ell^{(1)}$ $c^{(1)}$
$a^{(d+1)}$ $\ell^{(d+1)}$ $c^{(d+1)}$
$a^{(d+2)}$ $\ell^{(d+2)}$
$a^{(2d+1)}$ $\ell^{(2d+1)}$
$A(Z)$
$B(Z)$
$\mathbb{a}^{(d+2)}$ $\mathbb{b}^{(d+2)}$ $\mathbb{c}^{(d+2)}$
$\mathbb{a}^{(2d+1)}$ $\mathbb{b}^{(2d+1)}$ $\mathbb{c}^{(2d+1)}$
a,b,c
x y
x·y
x-a
y-b

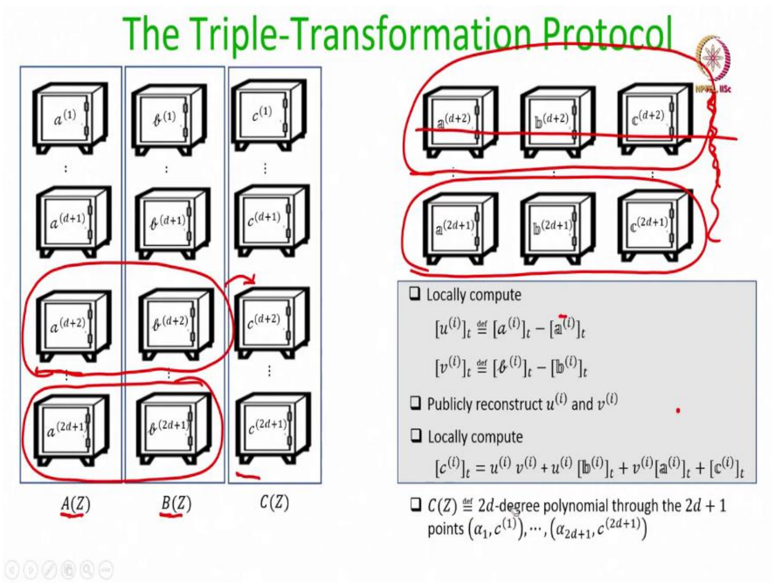
- ❑ For $i = d+2, \cdots, 2d+1$:
  - ❖ Apply Beaver's method on $([a^{(i)}]_t, [\ell^{(i)}]_t)$, using $([\mathbb{a}^{(i)}]_t, [\mathbb{b}^{(i)}]_t, [\mathbb{c}^{(i)}]_t)$ as the auxiliary triple

So, what was the Beaver's method? So, remember Beaver's method helps you to do the following. If you have a multiplication gate say $x\ AND\ y$ and you want to compute $x \cdot y$ and if you have an auxiliary secret shared say (a, b, c) in the pre processing phase or somehow you have got that secret share triplet (a, b, c); then we would have publicly computed $x - a$, we would have publicly computed $y - b$, and then we would have expressed the product of x and y as a linear function of secret shared a, b, c.

That was the Beaver’s method for computing the secret sharing of $x \cdot y$. What we are proposing here is to compute a secret sharing of the product of the a and b components of the last d output triplets by using the d number of secret shared triplets, input secret shared triplets we have not which we have not yet touched.

So, remember throughout this process of defining the A polynomial, B polynomial we have not touched the last d number of input secret share triplets. We are now going to utilize them for computing the remaining points on my C polynomial. So, what we will do now is the following.

(Refer Slide Time: 30:28)



The Triple-Transformation Protocol

- ❑ Locally compute

  $[u^{(i)}]_t \overset{\text{def}}{=} [a^{(i)}]_t - [\mathbb{a}^{(i)}]_t$

  $[v^{(i)}]_t \overset{\text{def}}{=} [b^{(i)}]_t - [\mathbb{b}^{(i)}]_t$

- ❑ Publicly reconstruct $u^{(i)}$ and $v^{(i)}$
- ❑ Locally compute

  $[c^{(i)}]_t = u^{(i)}\, v^{(i)} + u^{(i)}\, [\mathbb{b}^{(i)}]_t + v^{(i)}[\mathbb{a}^{(i)}]_t + [\mathbb{c}^{(i)}]_t$

- ❑ $C(Z) \overset{\text{def}}{=} 2d$-degree polynomial through the $2d+1$ points $(\alpha_1, c^{(1)}), \cdots, (\alpha_{2d+1}, c^{(2d+1)})$

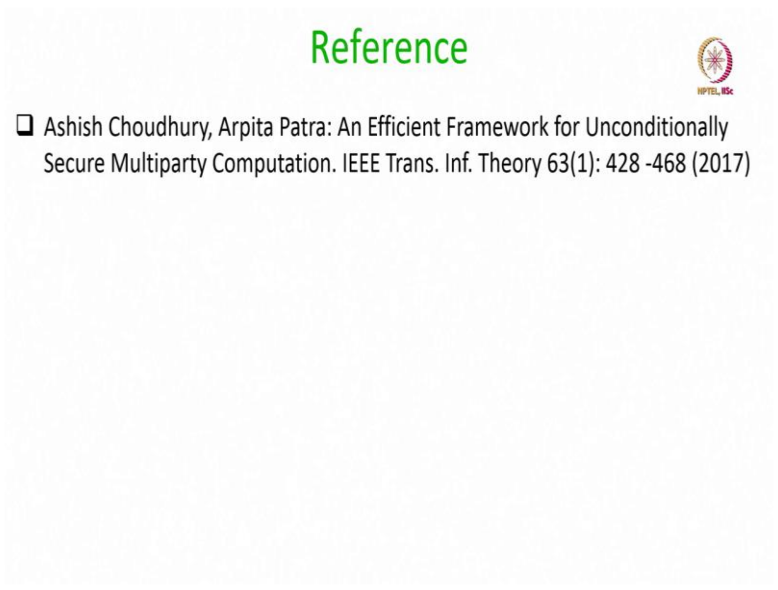
So, for instance, we take this pair of values, which are points on my A and B polynomials and now I want to compute a secret sharing of a times; I want to compute a secret sharing of this value. For that I will apply the Beaver’s method using this as my auxiliary triplet.

So, I will perform this operation and that will help me to give a secret sharing of an output value, which will be t shared and then I will simply ignore this input triplet, like that I will take this pair of values and I will take the last secret shared input triplet apply the Beaver’s method and then I will get a secret sharing of this value.

And now, I have $2d + 1$ number of secret shared c points, using which I will define my C polynomial. So, you can see the process of defining the C polynomial is different from the process of defining the A and B polynomials. For defining the A and B polynomials, we do not need any interaction whatsoever. First fix the $d+1$ points and extend them, non-interactively to get the remaining points.

Do the same thing for the B polynomial for the C polynomial half of the points. In fact, more than half of the points are fixed. And then the remaining points are computed interactively by applying the Beaver's method, using the input triplets the last d number of secret shared input triplets as the auxiliary triplets. So, that is a triple transformation process.

(Refer Slide Time: 32:35)



Reference

- Ashish Choudhury, Arpita Patra: An Efficient Framework for Unconditionally Secure Multiparty Computation. IEEE Trans. Inf. Theory 63(1): 428 -468 (2017)

In the next lecture we will discuss the properties achieved by our triple transformation process.

Thank you.